Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

про виконання лабораторної роботи № 8

з курсу:

«Спеціалізовані мови програмування»

**Виконала:**

студентка гр. ІТ-31

Ірина ПЕРХУН

**Прийняв:**

Сергій ЩЕРБАК

Львів 2023

**Тема:** Візуалізація та обробка даних за допомогою спеціалізованих бібліотек Python

**Мета роботи:** Розробка додатка для візуалізації CSV-наборів даних за допомогою Matplotlib та базових принципів ООП (наслідування, інкапсуляція, поліморфізм)

**Хід роботи**

*Завдання 1: Вибір CSV-набору даних*

*Оберіть CSV-набір даних, який ви хочете візуалізувати. Переконайтеся, що він містить відповідні дані для створення змістовних візуалізацій.*

*Завдання 2: Завантаження даних з CSV*

*Напишіть код для завантаження даних з CSV-файлу в ваш додаток Python. Використовуйте бібліотеки, такі як Pandas, для спрощення обробки даних.*

*Завдання 3: Дослідження даних*

*Визначте екстремальні значення по стовцям*

*Завдання 4: Вибір типів візуалізацій*

*Визначте, які типи візуалізацій підходять для представлення вибраних наборів даних. Зазвичай це може бути лінійні графіки, стовпчикові діаграми, діаграми розсіювання, гістограми та секторні діаграми.*

*Завдання 5: Підготовка даних*

*Попередньо обробіть набір даних за необхідністю для візуалізації. Це може включати виправлення даних, фільтрацію, агрегацію або трансформацію.*

*Завдання 6: Базова візуалізація*

*Створіть базову візуалізацію набору даних, щоб переконатися, що ви можете відображати дані правильно за допомогою Matplotlib. Розпочніть з простої діаграми для візуалізації однієї змінної.*

*Завдання 7: Розширені візуалізації*

*Реалізуйте більш складні візуалізації, виходячи з характеристик набору. Поекспериментуйте з різними функціями Matplotlib та налаштуваннями.*

*Завдання 8: Декілька піддіаграм*

*Навчіться створювати кілька піддіаграм в межах одного малюнка для відображення декількох візуалізацій поруч для кращого порівняння.*

*Завдання 9: Експорт і обмін*

*Реалізуйте функціональність для експорту візуалізацій як зображень (наприклад, PNG, SVG) або інтерактивних веб-додатків (наприклад, HTML)*

**data\_loader.py**

import pandas as pd

class DataLoader:

    """

    A utility class for loading data from CSV files using pandas.

    Methods:

    --------

    load\_csv(file\_path):

        Load data from a CSV file and return a pandas DataFrame.

    """

    @staticmethod

    def load\_csv(file\_path):

        """

        Load data from a CSV file and return a pandas DataFrame.

        Parameters:

        -----------

        file\_path (str): The path to the CSV file.

        Returns:

        --------

        pd.DataFrame: The loaded data.

        """

        return pd.read\_csv(file\_path)

**data\_visualizer\_gui.py**

from tkinter import filedialog

import tkinter as tk

from tkinter import ttk, messagebox

from matplotlib import pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import pandas as pd

from PIL import Image

from classes.lab8.data\_loader import DataLoader

from classes.lab8.plot\_generator import PlotGenerator

from classes.lab8.file\_exporter import FileExporter

class DataVisualizerGUI:

    def \_\_init\_\_(self, master, csv\_file):

        self.master = master

        self.master.title("Data Visualizer GUI")

        self.data = DataLoader.load\_csv(csv\_file)

        self.columns = list(self.data.columns)

        self.info\_label = None

        self.current\_plot = None

        self.export\_button = None

        self.plot\_types = [

            'Histogram (Age)', 'Bar (Department & Salary)', 'Scatter (Experience & Salary)',

            'Pie (Gender)', 'Line (Salary & Time)', 'Bar (Department Distribution)'

        ]

        self.plot\_type\_combobox = ttk.Combobox(self.master, values=self.plot\_types)

        show\_plot\_button = tk.Button(self.master, text="Show Plot", command=self.show\_selected\_plot)

        self.column\_combobox = ttk.Combobox(self.master, values=self.columns)

        show\_extremes\_button = tk.Button(self.master, text="Show Extremes", command=self.show\_extremes)

        self.plot\_container = tk.Frame(self.master)

        show\_multiple\_subplots\_button = tk.Button(self.master, text="Show Multiple Subplots", command=self.show\_multiple\_subplots)

        self.plot\_type\_combobox.pack(pady=5)

        show\_plot\_button.pack(pady=5)

        self.column\_combobox.pack(pady=5)

        show\_extremes\_button.pack(pady=5)

        show\_multiple\_subplots\_button.pack(pady=5)

        self.plot\_container.pack(pady=5)

        self.export\_button = tk.Button(self.master, text="Export Plot", command=self.export\_current\_plot\_as\_image)

    def show\_extremes(self):

        selected\_column = self.column\_combobox.get()

        if not selected\_column:

            messagebox.showinfo("Error", "Please select a column for analysis.")

            return

        if self.info\_label:

            self.info\_label.pack\_forget()

        for widget in self.plot\_container.winfo\_children():

            widget.destroy()

        if self.data[selected\_column].dtype == 'object':

            self.analyze\_text\_column(selected\_column)

        else:

            self.analyze\_numeric\_column(selected\_column)

    def analyze\_text\_column(self, selected\_column):

        if selected\_column == 'birth\_date':

            min\_date = self.data[selected\_column].min()

            max\_date = self.data[selected\_column].max()

            extremes = pd.Series({

                'Min Date': min\_date,

                'Max Date': max\_date

            })

            info\_text = f"Column Information: {selected\_column}\n"

            info\_text += f"Min Date: {min\_date}\n"

            info\_text += f"Max Date: {max\_date}"

            self.show\_info\_in\_gui(info\_text)

            self.plot\_extremes\_dates(extremes)

        else:

            self.analyze\_categorical\_column(selected\_column)

    def analyze\_numeric\_column(self, selected\_column):

        extremes = self.data[selected\_column].describe()

        self.plot\_extremes\_numeric(extremes)

    def analyze\_categorical\_column(self, selected\_column):

        unique\_values = len(self.data[selected\_column].unique())

        most\_common\_value = self.data[selected\_column].mode().iloc[0]

        most\_common\_count = self.data[selected\_column].value\_counts().iloc[0]

        extremes = pd.Series({

            'Unique Values': unique\_values,

            'Most Common Value': f"{most\_common\_value} (Count: {most\_common\_count})"

        })

        info\_text = f"Column Information: {selected\_column}\n"

        info\_text += f"Number of Unique Values: {unique\_values}\n"

        info\_text += f"Most Common Value: {most\_common\_value} (Count: {most\_common\_count})"

        self.show\_info\_in\_gui(info\_text)

    def show\_info\_in\_gui(self, info\_text):

        for widget in self.plot\_container.winfo\_children():

            widget.destroy()

        if self.export\_button:

            self.export\_button.destroy()

        self.info\_label = tk.Label(self.plot\_container, text=info\_text)

        self.info\_label.pack()

    def plot\_extremes\_dates(self, extremes):

        fig, ax = plt.subplots()

        min\_date = pd.to\_datetime(extremes['Min Date'])

        max\_date = pd.to\_datetime(extremes['Max Date'])

        ax.bar(['Min Date', 'Max Date'], [min\_date, max\_date])

        ax.set\_title("Extremes (Text Column)")

        ax.set\_ylabel("Date")

        ax.set\_xlabel("Attribute")

        self.update\_plot(fig)

    def plot\_extremes\_numeric(self, extremes):

        fig, ax = plt.subplots()

        extremes.plot(kind='bar', ax=ax)

        ax.set\_title("Extremes (Numeric Column)")

        ax.set\_ylabel("Value")

        ax.set\_xlabel("Attribute")

        self.update\_plot(fig)

    def update\_plot(self, fig):

        if self.current\_plot:

            self.current\_plot.get\_tk\_widget().forget()

        self.current\_plot = FigureCanvasTkAgg(fig, master=self.plot\_container)

        self.current\_plot.draw()

        self.current\_plot.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

        self.update\_export\_button\_state()

        plt.tight\_layout()

    def update\_export\_button\_state(self):

        if self.current\_plot:

            if self.export\_button:

                self.export\_button.destroy()

                self.export\_button = tk.Button(self.master, text="Export Plot", command=self.export\_current\_plot\_as\_image)

                self.export\_button.pack()

        else:

            if self.export\_button:

                self.export\_button.destroy()

    def show\_selected\_plot(self):

        selected\_plot\_type = self.plot\_type\_combobox.get()

        if not selected\_plot\_type:

            messagebox.showinfo("Error", "Please select a plot type.")

            return

        if self.info\_label:

            self.info\_label.pack\_forget()

        for widget in self.plot\_container.winfo\_children():

            widget.destroy()

        self.plot\_selected\_type(selected\_plot\_type)

    def plot\_selected\_type(self, selected\_plot\_type):

        if selected\_plot\_type == 'Histogram (Age)':

            self.plot\_histogram\_age()

        elif selected\_plot\_type == 'Bar (Department & Salary)':

            self.plot\_bar\_department\_salary()

        elif selected\_plot\_type == 'Scatter (Experience & Salary)':

            self.plot\_scatter\_experience\_salary()

        elif selected\_plot\_type == 'Pie (Gender)':

            self.plot\_pie\_gender()

        elif selected\_plot\_type == 'Line (Salary & Time)':

            self.plot\_line\_salary\_time()

        elif selected\_plot\_type == 'Bar (Department Distribution)':

            self.plot\_bar\_department\_distribution()

    def plot\_histogram\_age(self):

        PlotGenerator.generate\_histogram\_age(self.data, self)

    def plot\_bar\_department\_salary(self):

        PlotGenerator.generate\_bar\_department\_salary(self.data, self)

    def plot\_scatter\_experience\_salary(self):

        PlotGenerator.generate\_scatter\_experience\_salary(self.data, self)

    def plot\_pie\_gender(self):

        PlotGenerator.generate\_pie\_gender(self.data, self)

    def plot\_line\_salary\_time(self):

        PlotGenerator.generate\_line\_salary\_time(self.data, self)

    def plot\_bar\_department\_distribution(self):

        PlotGenerator.generate\_bar\_department\_distribution(self.data, self)

    def show\_multiple\_subplots(self):

        if self.info\_label:

            self.info\_label.pack\_forget()

        PlotGenerator.generate\_multiple\_subplots(self.data, self)

    def export\_current\_plot\_as\_image(self):

        if self.current\_plot:

            FileExporter.export\_plot\_as\_image(self.current\_plot)

            messagebox.showinfo("Success", "Plot successfully saved as an image.")

**file\_exporter.py**

from tkinter import filedialog, messagebox

class FileExporter:

    """Utility class for exporting Matplotlib plots as images."""

    @staticmethod

    def export\_plot\_as\_image(figure):

        """

        Export the given Matplotlib figure as an image.

        Parameters:

        - figure (matplotlib.figure.Figure): The Matplotlib figure to be exported.

        Returns:

        - None

        """

        file\_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "\*.png")])

        if file\_path:

            figure.savefig(file\_path)

            messagebox.showinfo("Success", "Plot successfully saved as an image.")

**main.py**

import os

import tkinter as tk

from classes.lab8.data\_visualizer\_gui import DataVisualizerGUI

class App:

    """

    Main application class for launching the Data Visualizer GUI.

    """

    def \_\_init\_\_(self, master):

        """

        Initialize the App.

        Parameters:

        - master (tk.Tk): The root Tkinter window.

        Returns:

        - None

        """

        self.master = master

        self.master.title("Main Application")

        # Get the absolute path to the current file

        lab\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

        csv\_file\_path = os.path.join(lab\_dir, 'employee\_data.csv')

        self.data\_visualizer = DataVisualizerGUI(self.master, csv\_file\_path)

    def run(self):

        """

        Run the main application loop.

        Returns:

        - None

        """

        self.master.mainloop()

def main():

    """

    Main function to create the Tkinter root window and run the application.

    Returns:

    - None

    """

    root = tk.Tk()

    app = App(root)

    app.run()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**plot\_generator.py**

from matplotlib import pyplot as plt

import pandas as pd

class PlotGenerator:

    """

    A class containing static methods to generate various types of plots.

    """

    @staticmethod

    def generate\_histogram\_age(data, gui\_instance):

        """

        Generate and display a histogram of employee ages.

        Parameters:

        - data (pd.DataFrame): The DataFrame containing employee data.

        - gui\_instance: An instance of the DataVisualizerGUI class.

        Returns:

        - None

        """

        fig, ax = plt.subplots()

        # Age histogram

        ax.hist(data['age'], bins=10, edgecolor='black', alpha=0.7)

        ax.set\_title("Histogram of Employee Ages")

        ax.set\_xlabel("Age")

        ax.set\_ylabel("Number of Employees")

        # Update the plot in the GUI container

        gui\_instance.update\_plot(fig)

    @staticmethod

    def generate\_bar\_department\_salary(data, gui\_instance):

        fig, ax = plt.subplots()

        department\_avg\_salary = data.groupby('department')['salary'].mean().sort\_values(ascending=False)

        department\_avg\_salary.plot(kind='bar', ax=ax, color='skyblue')

        ax.set\_xticklabels(department\_avg\_salary.index, rotation=45, ha="right")

        ax.set\_title("Average Salary by Departments")

        ax.set\_xlabel("Department")

        ax.set\_ylabel("Average Salary")

        gui\_instance.update\_plot(fig)

    @staticmethod

    def generate\_scatter\_experience\_salary(data, gui\_instance):

        fig, ax = plt.subplots()

        ax.scatter(data['experience'], data['salary'], color='green', alpha=0.7)

        ax.set\_title("Scatter Plot of Experience and Salary")

        ax.set\_xlabel("Experience (years)")

        ax.set\_ylabel("Salary")

        gui\_instance.update\_plot(fig)

    @staticmethod

    def generate\_pie\_gender(data, gui\_instance):

        fig, ax = plt.subplots()

        gender\_counts = data['gender'].value\_counts()

        gender\_counts.plot(kind='pie', autopct='%1.1f%%', colors=['lightcoral', 'lightblue'], ax=ax)

        ax.set\_title("Distribution by Gender")

        gui\_instance.update\_plot(fig)

    @staticmethod

    def generate\_line\_salary\_time(data, gui\_instance):

        fig, ax = plt.subplots()

        data['birth\_date'] = pd.to\_datetime(data['birth\_date'])

        salary\_time = data.groupby(data['birth\_date'].dt.year)['salary'].mean()

        salary\_time.plot(kind='line', marker='o', ax=ax, color='orange')

        ax.set\_title("Line Plot of Salary Changes Over Time")

        ax.set\_xlabel("Year")

        ax.set\_ylabel("Average Salary")

        gui\_instance.update\_plot(fig)

    @staticmethod

    def generate\_bar\_department\_distribution(data, gui\_instance):

        fig, ax = plt.subplots()

        department\_distribution = data['department'].value\_counts().sort\_values(ascending=False)

        department\_distribution.plot(kind='bar', ax=ax, color='salmon')

        ax.set\_xticklabels(department\_distribution.index, rotation=45, ha="right")

        ax.set\_title("Bar Chart\n(Distribution by Departments)")

        ax.set\_xlabel("Department")

        ax.set\_ylabel("Number of Employees")

        gui\_instance.update\_plot(fig)

    @staticmethod

    def generate\_multiple\_subplots(data, gui\_instance):

        """

        Generate and display multiple subplots.

        Parameters:

        - data (pd.DataFrame): The DataFrame containing employee data.

        - gui\_instance: An instance of the DataVisualizerGUI class.

        Returns:

        - None

        """

        if gui\_instance.info\_label:

            gui\_instance.info\_label.pack\_forget()

        fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(12, 6))

        axes[0, 0].hist(data['age'], bins=10, edgecolor='black', alpha=0.7)

        axes[0, 0].set\_title("Histogram of Employee Ages")

        axes[0, 1].scatter(data['age'], data['salary'], color='purple', alpha=0.7)

        axes[0, 1].set\_title("Scatter Plot\n(Age and Salary)")

        department\_avg\_salary = data.groupby('department')['salary'].mean().sort\_values(ascending=False)

        bar\_plot = department\_avg\_salary.plot(kind='bar', ax=axes[0, 2], color='skyblue')

        axes[0, 2].set\_title("Bar Chart\n(Average Salary by Departments)")

        bar\_plot.set\_xticklabels(bar\_plot.get\_xticklabels(), rotation=45, ha="right")

        gender\_counts = data['gender'].value\_counts()

        gender\_counts.plot(kind='pie', autopct='%1.1f%%', colors=['lightcoral', 'lightblue'], ax=axes[1, 0])

        axes[1, 0].set\_title("Pie Chart\n(Distribution by Gender)")

        department\_counts = data['department'].value\_counts()

        bar\_plot = department\_counts.plot(kind='bar', ax=axes[1, 1], color='salmon')

        axes[1, 1].set\_title("Bar Chart\n(Distribution by Departments)")

        bar\_plot.set\_xticklabels(bar\_plot.get\_xticklabels(), rotation=45, ha="right")

        data['birth\_date'] = pd.to\_datetime(data['birth\_date'])

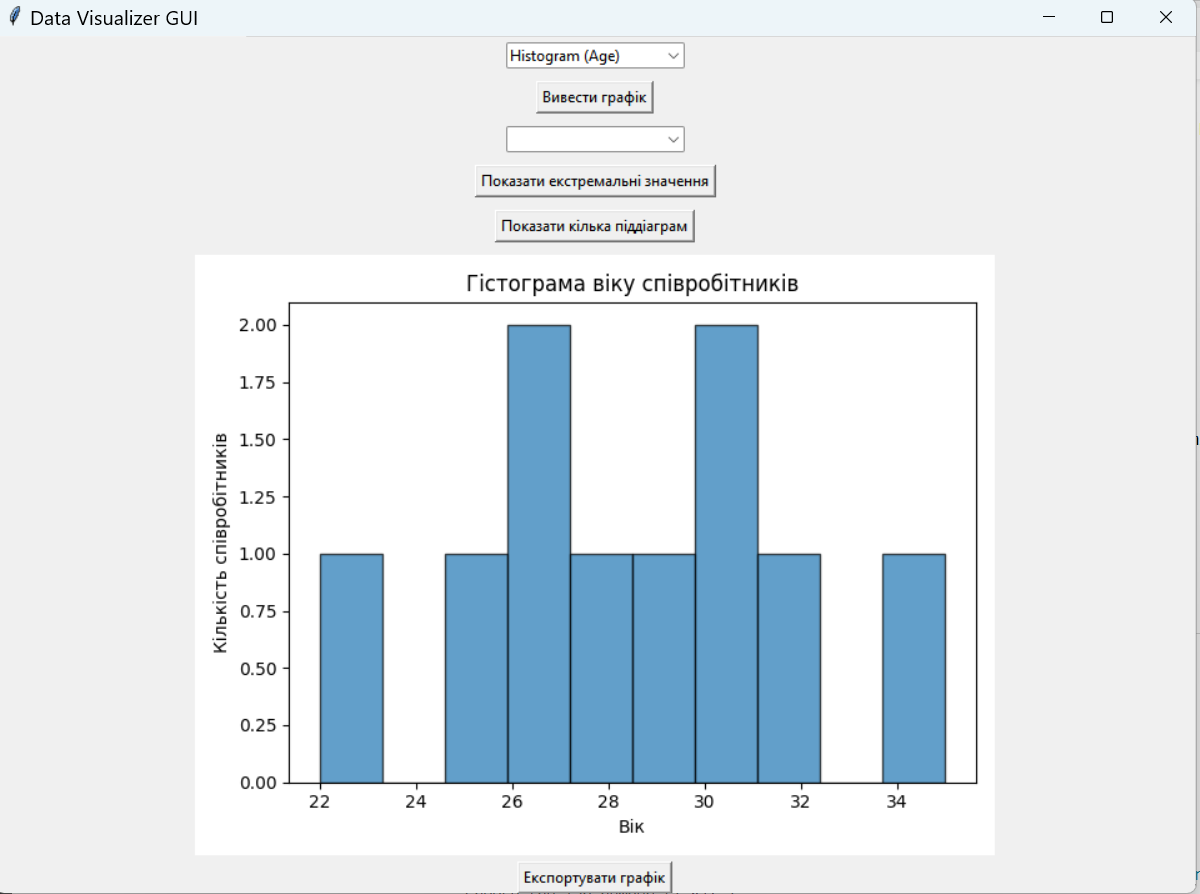
        salary\_time = data.groupby(data['birth\_date'].dt.year)['salary'].mean()

        salary\_time.plot(kind='line', marker='o', ax=axes[1, 2], color='orange')

        axes[1, 2].set\_title("Line Plot\n(Average Salary Over Time)")

        plt.tight\_layout()

        gui\_instance.update\_plot(fig)



*Рис. 1 – Результат виконання*

**Висновок:** на даній лабораторній роботі я розробила додаток для візуалізації CSV-наборів даних за допомогою Matplotlib та базових принципів ООП (наслідування, інкапсуляція, поліморфізм).